SANDIA REPORT

SAND2015-1118
Unlimited Release
Printed March 2015

Sandia Data Archive (SDA) file specifications

Daniel H. Dolan and Tom Ao

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2015-1118
Unlimited Release
Printed March 2015

Sandia Data Archive (SDA) file specifications

Daniel H. Dolan and Tom Ao
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1189

Abstract

The Sandia Data Archive (SDA) format is a specific implementation of the HDF5 (Hierarchal Data Format version 5) standard. The format was developed for storing data in a universally accessible manner. SDA files may contain one or more data records, each associated with a distinct text label. Primitive records provide basic data storage, while compound records support more elaborate grouping. External records allow text/binary files to be carried inside an archive and later recovered.

This report documents version 1.0 of the SDA standard. The information provided here is sufficient for reading from and writing to an archive. Although the format was original designed for use in MATLAB, broader use is encouraged.

Acknowledgments

Much of this work was inspired by the Portable File Format (PFF) created at Sandia by Dave Seidel. Standard alternatives to PFF (notably HDF5) have emerged over the last two decades, but the generality they provide can be overwhelming. The SDA format restricts some of the complexity of HDF5 while retaining portability and flexibility.

Many design aspects of SDA are based on the MATLAB functions `h5create`, `h5write`, `h5writeatt`, `h5read`, and `h5readatt`. Similar capabilities should be available in other computer languages. For more information about HDF5, consult the HDF5 home page at `http://www.hdfgroup.org/HDF5/`.

Contents

Figures

Tables

/ (root group)
Attributes: FileFormat, FormatVersion, Created, Updated, Writable
Dataset: (none)

/curly (record group)
Attributes, Datasets/Subgroups

/larry (record group)
Attributes, Datasets/Subgroups

/moe (record group)
Attributes, Datasets/Subgroups

Figure 1. Conceptual archive layout

1 Introduction

Various text and binary file formats are available for storing digital information. Text formats are flexible (and human-readable) but poorly suited for large datasets. Binary formats are more efficient than text, although flexibility and readability are vastly diminished. In either case, efficient storage of grouped information in a cross-platform manner is challenging.

Version 5 of the Hierarchical Data Format (HDF5)[1] standard provides a practical solution to these challenges. HDF5 is not a specific file format, but rather a model for information storage/management using datasets, groups, and attributes.

Datasets contain multi-dimensional arrays of arbitrary size.

Groups provides organizational structure for datasets.

Attributes describes the content of datasets and groups.

HDF5 libraries are available in virtually all modern computer languages, standardizing file read and write operations. HDF5 files are easier to navigate and access than custom binary files, combining the flexibility of a text format with the efficiency of a binary format.

A specific implementation of HDF5, the Sandia Data Archive (SDA), was developed for the SMASH (Sandia Matlab AnalysiS Hierarchy) toolbox.[2] SDA addresses the need for sharing groups of related information, known as records, across computer platforms. Although SDA originated as part of SMASH, the format is useful outside of the toolbox and MATLAB, providing efficient data sharing for many other applications.

This report provides detailed specifications for the SDA format. Section 2 provides a general format overview. Sections 3–5 describe the records used in an archives. Section 6 describes how records are read from and written to archives. A report summary and future work are summarized in Section 7.

2 Format overview

SDA files are based on and entirely compatible with HDF5. Virtually all modern computer languages have libraries for reading and writing HDF5 files, so adapting these routines to read/write SDA is straightforward. For clarity, archives use the *.sda extension instead of the standard *.h5 extension. Figure 1 shows a conceptual layout for an archive file.

Table 1. Root group attributes

Attribute	Description	Value
FileFormat	Format verification	'SDA'
FormatVersion	Format verification	'1.0'
Writable	SMASH write protection	'yes' or 'no'
Created	Date/time string	(DD-MMM-YYYY HH:MM:SS)
Updated	Date/time string	(DD-MMM-YYYY HH:MM:SS)

Table 2. Allowed record types

Type	Comments	
numeric	Numeric arrays of arbitrary size/dimension	primitive
logical	Logical arrays of arbitrary size/dimension	
character	Character arrays of arbitrary size/dimension	
function	MATLAB function handles	
cell	Cell arrays of arbitrary size/dimension	
structure	Structured data	compound
object	Custom MATLAB object	
file	File stored inside an archive	external
split	File split across multiple archives	

The root group `/` is the starting point for the every archive. This group has the attributes shown in Table 1; no datasets are associated with the root group. Root attributes are defined at file creation, and some (Created, FileFormat, FormatVersion) are never changed. The Writable attribute specifies if new records can be written to the archive or if existing records can be modified. The Updated attribute is changed whenever records are added or modified.

Archive records are located in distinct base groups directly under the root group. Records have a unique text label that defines the base group name. *Repeated labels are not allowed!* In Figure 1, subgroups `/larry`, `/curly`, and `/moe` correspond to record labels 'larry', 'curly', and 'moe' (respectively). Labels can use any ASCII character except forward/backward slashes and are treated in a case-sensitive manner. Although there are no length restrictions, labels should be a short as possible without being overly cryptic. Record access is managed exclusively through these labels. Numerical indices are never assigned!

Table 2 summarizes the record types that may be found in an archive. Record types fall into three categories—primitive, compound, and external—described in the following sections.

3 Primitive records

Primitive records consist of a base group with a single dataset. The dataset repeats the group name as its own: a primitive record named 'larry' is stored as the base group `/larry` containing the dataset `/larry/larry`. Attributes are assigned to the base group and dataset to manage data insertion and extraction. Table 3 lists the base group attributes for a primitive record, some of which (RecordType and Empty) are repeated as dataset attributes.

SDA recognizes four primitive record types: numeric, logical, character, and function. The first three types provide general purpose array storage, while the fourth type is specific to MATLAB. Figure 2 shows

Table 3. Primitive record attributes

Attribute	Description	Value
RecordType*	Record type	numeric/logical/character/function
Empty*	Indicates empty records	'yes' or 'no'
Description	Optional record description	(string)
Deflate	Lossless compression level	0–9 (default is 0)
Command†	MATLAB function name	(string)

* attribute repeated in dataset

† function records only

an archive containing two numeric records (double and single precision representations of a 3×4 array), an empty character record, a logical record (true), and a function record (sine).

3.1 Numeric, character, and logical records

Numeric, character, and logical records store an array of arbitrary size and dimensionality in the group's sole dataset. The stored array type is:

- Double, single, integer (8/16/32/64-bit), or unsigned integer (8/16/32/64-bit) for numeric records, depending on the source variable.
- 8-bit unsigned integers for character records (ASCII conversion).
- 8-bit unsigned integers for for logical records (true=1, false=0).

Stored arrays have fixed dimensionality but arbitrary size. For example, the 'curly' record in Figure 2 has a 3×4 dataset with a maximum size of Inf$\times$ Inf. This dataset is constrained to be two-dimensional, but the size can be expanded as needed along either dimension.

Empty records are indicated by an Empty attribute of 'yes'. Since HDF5 does not permit truly empty datasets, a single NaN value is used as a placeholder. Placeholder datasets should be ignored when a record is defined to be empty.

SDA datasets always specify a chunk size, which is used in lossless compression (deflation) and allows dataset expansion. Chunk sizes can be set to match the stored array size (as in Figure 2) or a subdomain of the array. Chunk sizes must be consistent with the dataset's dimensionality, *e.g.* a scalar chunk size for a two-dimensional array is 1×1, not 1. Although certain datasets may have an optimal chunk size, the actual information stored/recovered in an archive does not depend on the chunk size setting. For more information about chunking and deflation, refer to the HDF5 standard.[1]

3.2 Function records

Function handles are a MATLAB data type for passing functions to other functions (integration, optimization, etc.). The inner workings of function handles are poorly documented. For archive storage, function handles are converted to an intermediate binary file (MAT format version 7.3) containing the calling sequence and all supporting data. This file is read into MATLAB as unsigned 8-bit integers, inserted as a numeric record, and reclassified as a function record.

This record type should not be used outside of MATLAB.

```
Group '/'
    Attributes:
        'FileFormat':  'SDA'
        'FormatVersion':  '1.0'
        'Created':  '10-Feb-2015 09:47:52'
        'Writable':  'yes'
        'Updated':  '10-Feb-2015 09:47:54'
    Group '/curly'
        Attributes:
            'RecordType':  'numeric'
            'Empty':  'no'
            'Deflate':  0.000000
            'Description':  '3x4 array (double)'
        Dataset 'curly'
            Size:  3x4
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F64LE (double)
            ChunkSize:  3x4
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
    Group '/curly 2'
        Attributes:
            'RecordType':  'numeric'
            'Empty':  'no'
            'Deflate':  0.000000
            'Description':  'Single version of curly'
        Dataset 'curly 2'
            Size:  3x4
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F32LE (single)
            ChunkSize:  3x4
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
    Group '/larry'
        Attributes:
            'RecordType':  'character'
            'Empty':  'yes'
            'Deflate':  0.000000
            'Description':  'An empty character array'
        Dataset 'larry'
            Size:  1x1
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x1
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'yes'
    Group '/moe'
        Attributes:
            'RecordType':  'logical'
            'Empty':  'no'
            'Deflate':  0.000000
            'Description':  'A 1x1 logical array'
        Dataset 'moe'
            Size:  1x1
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x1
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'logical'
                'Empty':  'no'
    Group '/my function'
        Attributes:
            'RecordType':  'function'
            'Empty':  'false'
            'Deflate':  0.000000
            'Description':  'Sine function'
        Dataset 'my function'
            Size:  1x10280
            MaxSize:  1x10280
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  []
            Filters:  none
            FillValue:  0
            Attributes:
                'RecordType':  'function'
                'Command':  'sin'
                'Empty':  'no'
```

Figure 2. Archive containing five primitive records

Table 4. Composite record attributes

Attribute	Description	Value
RecordType	Record type	cell/structure/object
Empty	Indicates empty records	'yes' or 'no'
Description	Optional record description	(string)
Deflate	Lossless compression level	0–9 (default is 0)
RecordSize*	Cell array size	(set of numbers)
FieldNames†	Field/property names	(string)
Class‡	Object class	(string)

* cell records only
† structure/object records only
‡ object records only

4 Composite records

Related information can be linked in an archive with a composite record. Composite records are stored in base groups, but with multiple datasets and subgroups (as needed). Table 4 lists the base group attributes for composite records. Many of these attributes are similar to those found in primitive groups; none are repeated as dataset attributes. Some attributes (RecordSize, FieldNames, Class) are specific to particular record types

SDA recognizes three composite record types: cell, structure, and object. Each record is motivated by a MATLAB variable type. Cell records are based on cell arrays, which organize data by numeric index. Structure records are based on structures, which use named fields instead of numeric indices. Object records are based on custom MATLAB objects, which SDA manages as a modified version of a structure record.

Figure 3 shows an archive containing a cell record ('bob') and a structure record ('dave'). Both records include a 3×4 numeric table and a set of text choices ('yes' or 'no'). These choices are defined in a nested cell array, and the structure record also contains a nested structure. Arbitrary nesting of cell arrays, structures, and objects is permitted.

4.1 Cell records

Cell records can store arrays of arbitrary size and dimensionality. Variable type, dimensionality, or size can vary between elements of this array. Every array element is assigned an integer index that increases along the sequential dimensions. For example, the k-th element of a 3×4 cell array is given by indices (m, n) as show below.

k	(m, n)
1	$(1, 1)$
2	$(2, 1)$
3	$(3, 1)$
4	$(1, 2)$
5	$(2, 2)$
$\vdots$	$\vdots$
12	$(3, 4)$

Each array element is stored as a dataset (numeric/character/logical/function data) or a subgroup (cell/structure/object data) of the parent group. Dataset/subgroup names are generated as 'element (number)'.

```
Group '/'
    Attributes:
        'FileFormat':  'SDA'
        'FormatVersion':  '1.0'
        'Created':  '10-Feb-2015 09:47:56'
        'Writable':  'yes'
        'Updated':  '10-Feb-2015 09:47:56'
    Group '/bob'
        Attributes:
            'RecordType':  'cell'
            'RecordSize':  1.000000 2.000000
            'Empty':  'no'
            'Deflate':  0.000000
            'Description':  'Nested cell array example'
        Dataset 'element 1'
            Size:  3x4
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F64LE (double)
            ChunkSize:  3x4
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
        Group '/bob/element 2'
            Attributes:
                'RecordType':  'cell'
                'RecordSize':  1.000000 2.000000
                'Empty':  'no'
            Dataset 'element 1'
                Size:  1x3
                MaxSize:  InfxInf
                Datatype:   H5T_STD_U8LE (uint8)
                ChunkSize:  1x3
                Filters:  deflate(0)
                FillValue:  0
                Attributes:
                    'RecordType':  'character'
                    'Empty':  'no'
            Dataset 'element 2'
                Size:  1x2
                MaxSize:  InfxInf
                Datatype:   H5T_STD_U8LE (uint8)
                ChunkSize:  1x2
                Filters:  deflate(0)
                FillValue:  0
                Attributes:
                    'RecordType':  'character'
                    'Empty':  'no'
    Group '/dave'
        Attributes:
            'RecordType':  'structure'
            'Empty':  'no'
            'FieldNames':  'table choices parameter '
            'Deflate':  0.000000
            'Description':  'Nested structure array example'
        Dataset 'table'
            Size:  3x4
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F64LE (double)
            ChunkSize:  3x4
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
    Group '/dave/choices'
        Attributes:
            'RecordType':  'cell'
            'RecordSize':  1.000000 2.000000
            'Empty':  'no'
        Dataset 'element 1'
            Size:  1x3
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x3
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
        Dataset 'element 2'
            Size:  1x2
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x2
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
    Group '/dave/parameter'
        Attributes:
            'RecordType':  'structure'
            'Empty':  'no'
            'FieldNames':  'offset amplitude '
        Dataset 'amplitude'
            Size:  1x1
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F64LE (double)
            ChunkSize:  1x1
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
        Dataset 'offset'
            Size:  1x1
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F64LE (double)
            ChunkSize:  1x1
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
```

Figure 3. Archive containing composite records

Table 5. External record attributes

Attribute	Description	Value
RecordType	Record type	cell/structure/object
Empty	Indicates empty records	'no'
Description	Record description	(string)
Deflate	Lossless compression level	0–9 (default is 0)
FieldNames*	Field names	(string)

* split records only

Datasets inside a cell record (nested or not) are identical to primitive record datasets but do not share RecordType or Empty attributes with the parent group.

4.2 Structure records

Structure records can store an arbitrary number of named fields containing primitive/compound values. Field names must adhere to MATLAB variable naming requirements: the first character is a letter and all other characters are letters/numbers/underscores (no white space allowed). Primitive data inside the record is stored in datasets of the base group, and composite data is stored using subgroups. Dataset/subgroup labels are generated from the field name.

4.3 Object records

Object records store custom MATLAB objects. Because objects are conceptually similar to structures, SDA uses a modified structure record for handling objects. Object properties are converted to structure fields, but otherwise the two record types are identical aside from the RecordType and Class attribute. The latter indicates the MATLAB class associated with the stored object.

Custom MATLAB objects present special challenges.

- The SMASH toolbox converts objects to a structures for SDA storage, but hidden/protected/private properties of the class may be omitted in the process. SMASH classes provide a "store" method to avoid this problem, but other classes may not do the same.

- SMASH attempts to restore archived objects based on the Class attribute. If the class is not available, the record is returned as a structure.

- Archived objects with linked properties may be incorrectly restored because of differences in property assignment order. SMASH looks for a static "restore" method when loading archived objects to manage property links. If this method is not provided, properties are loaded in the order they are received (usually alphabetical).

5 External records

External records allow text and binary data files to be loaded into an archive. Table 5 lists the base group attributes for external records, which are similar to those found in numeric/structure records.

SDA recognizes two external record types: file and split. Figure 4–5 shows archives containing all or part of a scanned image file 'Otter.png' (55,772 bytes) using file and split records.

```
Group '/'
    Attributes:
        'FileFormat':  'SDA'
        'FormatVersion':  '1.0'
        'Created':  '10-Feb-2015 09:47:57'
        'Writable':  'yes'
        'Updated':  '10-Feb-2015 09:47:58'
    Group '/Otter.png'
        Attributes:
            'Empty':  'no'
            'Deflate':  0.000000
            'RecordType':  'file'
            'Description':  'Scanned graphic'
        Dataset 'Otter.png'
            Size:  1x55772
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x55772
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
    Group '/curly'
        Attributes:
            'RecordType':  'numeric'
            'Empty':  'no'
            'Deflate':  0.000000
            'Description':  'A 3x4 array'
        Dataset 'curly'
            Size:  3x4
            MaxSize:  InfxInf
            Datatype:   H5T_IEEE_F64LE (double)
            ChunkSize:  3x4
            Filters:  deflate(0)
            FillValue:  0.000000
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
```

Figure 4. Archive containing a file record

5.1 File records

File records store individual text/binary files for supplemental documentation (notes, spreadsheets, etc.) of the archive's contents. Bytes from the source file are stored as a sequence of 8-bit integers in a modified numeric record: the base group's RecordType attribute is changed from numeric to file and named after the source file (including the extension). File records can coexist with primitive and compound records.

For the example shown in Figure 4, all 55,772 bytes of the file 'Otter.png' are written to the file record `/Otter.png`. File records always store the entire source file in an archive, regardless of size. Using the Deflate attribute, lossless compression can be applied when the record is created; changing the Deflate attribute afterwards has no effect.

5.2 Split records

Split records allow individual files to be divided across multiple archive files. Each file contains a portion of the source file as reclassified structure record with three fields.

Bytes indicates the number of source file bytes stored in the archive file.

OriginalName indicates the source file name.

SplitFiles lists the name of all archive files the source was split into.

Each split record resides in its own archive file in a base group named `/split file`. Archives containing split records are named after the source file and their split position separated by two underscore characters. For example, the file `Otter.png` split into archive files `Otter.png__file1.sda` and `Otter.png__file2.sda` in Figure 5.

Split records are intended for size-limited transmission, such as electronic email. A maximum number of bytes are written to each split record, using as many files as needed to span the bytes of the source file. For the example in Figure 5, the maximum number of written bytes is 30,000. The first archive uses precisely this number of bytes, while the remaining 25,772 bytes are written to the second archive. As with file records, the Deflate attribute can be used to apply lossless compression during record creation only.

Archive file `Otter.png__file1.sda`**:**

```
Group '/'
    Attributes:
        'FileFormat':  'SDA'
        'FormatVersion':  '1.0'
        'Created':  '10-Feb-2015 09:47:59'
        'Updated':  '10-Feb-2015 09:47:59'
        'Writable':  'no'
    Group '/split file'
        Attributes:
            'Empty':  'no'
            'FieldNames':  'Bytes OriginalName SplitFiles '
            'Description':  'File 1 of 2'
            'Deflate':  0.000000
            'RecordType':  'split'
        Dataset 'Bytes'
            Size:  30000x1
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  30000x1
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
        Dataset 'OriginalName'
            Size:  1x9
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x9
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
    Group '/split file/SplitFiles'
        Attributes:
            'RecordType':  'cell'
            'RecordSize':  1.000000 2.000000
            'Empty':  'no'
        Dataset 'element 1'
            Size:  1x87
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x87
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
        Dataset 'element 2'
            Size:  1x87
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x87
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
```

Archive file `Otter.png__file2.sda`**:**

```
Group '/'
    Attributes:
        'FileFormat':  'SDA'
        'FormatVersion':  '1.0'
        'Created':  '10-Feb-2015 09:47:59'
        'Updated':  '10-Feb-2015 09:48:00'
        'Writable':  'no'
    Group '/split file'
        Attributes:
            'Empty':  'no'
            'FieldNames':  'Bytes OriginalName SplitFiles '
            'Description':  'File 2 of 2'
            'Deflate':  0.000000
            'RecordType':  'split'
        Dataset 'Bytes'
            Size:  25772x1
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  25772x1
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'numeric'
                'Empty':  'no'
        Dataset 'OriginalName'
            Size:  1x9
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x9
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
    Group '/split file/SplitFiles'
        Attributes:
            'RecordType':  'cell'
            'RecordSize':  1.000000 2.000000
            'Empty':  'no'
        Dataset 'element 1'
            Size:  1x87
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x87
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
        Dataset 'element 2'
            Size:  1x87
            MaxSize:  InfxInf
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1x87
            Filters:  deflate(0)
            FillValue:  0
            Attributes:
                'RecordType':  'character'
                'Empty':  'no'
```

Figure 5. Split file example

6 Accessing archives

Archive files always use the .sda extension (case insensitive), and the root group attribute FileFormat must be 'SDA'. The FormatVersion attribute is currently '1.0', but this may change in the future. All low-level read/write operations should be performed with HDF5 library functions, not user-written routines!

Reading archive records depends on the RecordType attribute.

- Primitive records have a single dataset that can read directly with HDF5 library functions unless the group's Empty attribute is 'yes'. Numeric records are equivalent to the stored dataset. ASCII conversion of the dataset is needed for character records; for logical records, dataset values of 0 and 1 must be converted to true and false, respectively.

  Function record datasets should be written as unsigned 8-bit integers to a temporary MATLAB *.MAT file. The command:

  ```
  >> load(tempfile,'data','-mat'); % MATLAB load function
  ```

  places the function handle in the workspace variable 'data', after which point the temporary file can be deleted. Function records are specific to MATLAB and should not be read in other languages.

- Composite records may have multiple datasets and subgroups. Datasets of the base group are numbered in cell records and named in structure/object records. Subgroups denote nesting of another composite record, which may have its own datasets as well as additional subgroups.

  Recursive analysis is helpful for navigating through a composite group. Starting from the base group, one moves down nested subgroups until only primitive records are found. Except for the group attributes Empty and RecordType, nested primitive records are identical to independent primitive record and are read in the same fashion. Dataset attributes always trump subgroup attributes.

  Cell record elements are stored by element number. Based on the RecordSize attribute, reshaping may be needed to place elements in the correct array location. Structure field and object properties are stored by name. Object arrays can be treated like structure arrays, but the full capabilities of these records are not available outside of MATLAB.

- External records store bytes for a file in two different ways. File records are essentially numeric records named after the stored file and may coexist with other records in an archive. Split records use a structure records containing a portion of the source file across multiple archives, each containing a single record. Source files may be reconstituted from either record type.

Writing records to new/existing archives involves reversing the read operations described above. However, there a several additional details to be managed.

- The root group attributes (FileFormat, FormatVersion, Created, Updated, and Writable) must be defined if they do not already exist. If the Writable attribute is 'no', modifications to the file should not be performed. When modifications are mode, the Updated attribute should be changed to the current date and time.

- New records must not use the same label as any existing record. *Do not overwrite an existing records with a record having the same label!*

- Split record archives always have a single record! Additional records should never be written to these archives.

Existing records can be modified with care.

- Attributes can be modified with varying levels of risk. For example, Description attributes are not required for record interpretation, so there is no risk in changing their value. Other attributes, such as FileFormat (root group) or RecordType (base groups) have very specific meaning and should not be modified. Refer to Table 1 and Tables 3–5 before modifying root and base group attributes, respectively. Dataset attributes (RecordType and Empty) should be modified with the same caution.
- Value replacement within a dataset's current size is always allowed. Datasets can be appended with array of consistent size: a 3×4 dataset can be expanded vertically with a $M \times 4$ array or horizontally with a $3 \times N$ array. Due to a limitation in HDF5, dataset size cannot be reduced directly.

HDF5 does not provide a direct way of deleting records. Instead, records must be copied to a new archive (omitting the "deleted" records) that replaces the existing archive.

7 Summary and future work

The SDA format stores labeled records as base groups in a renamed HDF5 file. Record labels are ASCII strings (without slashes) of any size. Records are accessed by label, so these strings should be descriptive and unique without becoming overly long. Full descriptions can be stored in the Description attribute associated with each record.

Archives support primitive, compound, and external records. Primitive records store a single type of information, such as an array (numeric, logical, and character records) or MATLAB function handle (function records). Compound records store grouped information with a numeric index (cell records) or by name (structure/object records). External records store entire files (file record) or portions of files (split record) for later reconstruction. With the exception of split records, which always reside on their own archive, archives may contain any number of primitive, compound, and external records.

The documentation provided here is sufficient to read archive files created by the SMASH toolbox. The format does not require the toolbox or MATLAB, and wider adoption is encouraged. Externally generated SDA files can pass information between applications, but whether this happens outside of SMASH remains to be seen. Anyone interested in expanding SDA support should contact the authors.

Version 1.0 of the SDA specification is quite broad, but a few capabilities were omitted. Structure arrays are supported in MATLAB but are not widely used outside of the `dir` function. Object arrays are also possible, though this can lead to unexpected behavior unless the methods of the class expect array input. Structure and object arrays require a numeric index and a field/property name, which can be implemented via nesting but is potentially confusing. At some point, structure/object arrays can be implemented in SDA, perhaps as "structures" and "objects" records. A workaround for the SDA version 1.0 is to place individual structures/objects inside cell arrays.

References

[1] The HDF Group. Hierarchical data format, version 5, (1997–2014). `http://www.hdfgroup.org/HDF5/`.

[2] D.H. Dolan and T. Ao. The Sandia Matlab AnalysiS Hierarchy (SMASH) package. Technical Report (in preparation), Sandia National Laboratories.

DISTRIBUTION:

1	MS 1106	T. Ao, 1646
1	MS 1134	D. Dalton, 6634
1	MS 1189	K. Cochrane, 1641
1	MS 1189	J.-P. Davis, 1646
1	MS 1189	D. Dolan, 1646
1	MS 1190	D. Bliss, 1675
1	MS 1193	P. Knapp, 1688
1	MS 1193	R. McBride, 1688
1	MS 1195	J. Brown, 1646
1	MS 1195	M. Furnish, 1646
1	MS 1454	B. Jilek, 2554
1	MS 1455	R. Wixom, 2554
1	MS 0899	Technical Library, 9536 (electronic copy)


Sandia National Laboratories